REMARKS/ARGUMENTS

Claims 20, 27, and 53 have been amended by inserting the word "to" between "due a" to correct an obvious typographical error.

1. Reply to the Rejection under 35 U.S.C. 101.

In paragraph 4 on page 3 of the Official Action, claims 1-28, 32, and 62-73 were rejected under 35 U.S.C. 101 as directed to non-statutory subject matter. Apparently these claims are being construed in such a way that the claimed method is considered not (1) tied to another statutory class (such as a particular apparatus), and not to (2) transform underlying subject matter such as an article or materials) to a different state or thing. The Official Action refers generally to Supreme Court precedent, recent Federal Circuit decisions, and Office's guidance to examiners, for a proposition of law that a 101 process must (1) be tied to another statutory class (such as a particular apparatus) or (2) to transform underlying subject matter (such as an article or materials) to a different state or thing. Upon using the search function at the "uspto.gov" web site, the applicants located a copy of a May 15, 2008 Memo from John J. Love, Deputy Commissioner for Patent Examination Policy, to Technology Center Directors, regarding In re Bilski, 545 F.3d 943, 88 USPQ2d 1385 (Fed Cir. 1008)(en banc)("reaffirm[ing] that the machine-or-transformation test, properly applied, is the governing test for determining patent eligibility of a process under § 101").

Please recall that applicants have already explained why they consider that their claimed invention is patentable subject matter under Supreme Court precedent and Federal Circuit decisions. See pages 26 to 37 of applicants' Amendment in Reply to Official Action filed Aug.

17, 2007. Applicants have already asserted that a "network file server" is a particular kind of machine, and when responding to a request from a network client to write to a file, the file server "transforms" the file from an original consistent state to a new consistent state. (Id., at 29.) The applicants' claims do not preempt a mathematical algorithm, abstract idea, mental steps, method of doing business, or fundamental principle of nature. (Cf. Gottschalk v. Benson, 409 U.S. 63 (1972)(the machine-or-transformation test is different from preemption of a mathematical algorithm). The applicants' method steps relate to internal operations of a specific kind of digital computer apparatus, namely a network file server. The applicants' claimed method provides more efficient concurrent access of network clients to shared files in a network file server, but the applicants' method claims do not "preempt" concurrent access of network clients to shared files in a network file server.

In view of the May 15, 2008 Memo from John J. Love, Deputy Commissioner for Patent Examination Policy, to Technology Center Directors, the applicants' method claims have been amended to further recite "network file server computer," and to further recite "file in data storage." Support for the applicants' "network file server" being a computer is found, for example, in applicants' FIG. 1 and in applicants' specification, page 11 lines 3-8. Applicants also respectfully submit that in the context of applicants' original claims as a whole, a person of ordinary skill would have understood that the recited network file server is a computer by definition. See, for example, the definition of "server" on the enclosed page 698 of Kaplan, Wiley Electrical and Electronics Engineering Dictionary, IEEE Press, 2004, John Wiley & Sons, Hoboken, NJ. Support for the file being in data storage is found, for example, in applicants' FIG. 2 (file system 54 in cached disk array 29) as described in applicants' specification on page

12 lines 17-23 and page 13 line 8 to page 14 line 5. Some of applicants' original dependent claims (claim 17 et seq.) further specify that the claimed file or file system is in disk storage, which is a common form of data storage. See also the definition of "data storage" on the enclosed page 168 of Kaplan, Wiley Electrical and Electronics Engineering Dictionary, IEEE Press, 2004, John Wiley & Sons, Hoboken, NJ.

Applicants respectfully submit that their method claims, as amended, are now tied to another statutory class, namely, a particular apparatus known as a "network file server computer," because the amended method claims positively recite "the method comprising the network file server computer responding to a concurrent write request from a client by: ..." Thus, the network file server computer is a particular apparatus, and not a human being, printed matter, or abstract idea, or some other non-statutory class. In addition, the network file server is performing the recited method steps in response to a specific event, namely, in response to a concurrent write request from a client.

Applicants also respectfully submit that their method claims, as amended, transform underlying subject matter (such as an article or materials) to a different state or thing. In particular, the file in data storage is underlying material that is transformed to a different state by the positively recited step of "(f) committing the metadata block to the file in the data storage; ..." The data storage itself also is underlying material that is transformed to a different state by the positively recited step of "(f) committing the metadata block to the file in the data storage; ..." The file in data storage is not an abstract idea, a disembodied collection of information, nor a signal in the ether. The file in data storage contains certain blocks of the data storage and data and metadata stored in the certain blocks of the data storage on a permanent or semi-permanent

basis on a data storage device or medium. Thus, the act of committing the metadata block to the file in the data storage changes the organization of the file in the data storage and also changes the physical state of the data storage device or medium, because the metadata block becomes a new part of the file in the data storage. See, for example, In re Bernhart, 417 F.2d 1395, 1400, 163 U.S.P.Q. 611, 616 (C.C.P.A. 1969) ("if a machine a machine is programmed in a new and unobvious way, it is physically different from the machine without that program; its memory elements are differently arranged.") See also WMS Gaming., Inc., v. International Game Technology, 184 F.3d 1339, 1348 (Fed. Cir. 1999)("A general purpose computer, or microprocessor, programmed to carry out an algorithm creates 'a new machine, because a general purpose computer in effect becomes a special purpose computer once it is programmed to perform particular functions pursuant to instructions from program software.' (citations omitted)). The transformation of the organization of the file in the data storage is central to the applicants' method claims. The applicants' method steps ensure consistency of the re-organization of the file in the data storage during concurrent write access to the file.

2. Reply to the Rejection under 35 U.S.C. 102(e).

In paragraph 6 on page 4 of the Official Action, claims 1-4, 10, 11, 15, 22, 32, 33-36, 44, 45, 49, 58, and 61-73 were rejected under 35 U.S.C. 102(e) as being anticipated by Chang et al. U.S. 2005/0039049 A1. Applicants respectfully traverse. Applicants respectfully submit that they made their invention prior to the effective date of Chang et al. Applicants also respectfully

submit that Chang et al. does not identically show every element of claims 1-4, 10, 11, 15, 22, 32, 33-36, 44, 45, 49, 58, and 61-73.

2a. Applicants' made their invention before the effective date of Chang et al.

Please find submitted herewith a Rule 131 Declaration of Sorin Faibish, and a Rule 131 Declaration of Richard C. Auchterlonie. The effective date of Chang et al. is its filing date of Aug. 14, 2003, which is less than six weeks before the applicants' filing date of Sept. 23, 2003. Pursuant to MPEP 2138.06, the enclosed Rule 131 Declarations evidence that on August 8, 2003, the undersigned patent attorney, Richard C. Auchterlonie, who prepared and filed the applicants' patent application, received a "red-lined" version of a first draft of the applicants' patent application from one of the inventors, Sorin Faibish. This "red-lined" version of the first draft of the applicants' patent application is evidence of a complete conception of the applicants' invention of the pending claims by August 8, 2003. The enclosed Rule 131 Declarations also evidence reasonable diligence from the August 14, 2003 effective date of Chang et al. until the constructive reduction to practice of the invention by the filing of the applicants' patent application on Sept. 23, 2003.

The enclosed Rule 131 Declarations evidence that from August 8, 2003 to Sept. 4, 2003, the undersigned patent attorney prepared a second draft of the applicants' patent application from the "red-lined" version of the first draft, and on Sept. 4, 2003, the second draft was sent to and received by the inventors on Sept. 4, 2003 for review and approval prior to filing. This second draft was reviewed and approved by the inventors for filing, and was filed with the Patent and Trademark Office on Sept. 23, 2003. Moreover, the "file wrapper" of the applicants' patent

application Ser. 10/668,467 should include an inventor's Declaration filed with the patent application on Sept. 23, 2003, and signed by the inventors Sachin Mullick on Sept. 23, 2003, Xiaoye Jiang on Sep. 23, 2003, and Peter Bixby on Sept 23, 2003. The diligence of 35 U.S.C. 102(g) relates to reasonable "attorney-diligence" and "engineering-diligence" (Keizer v. Bradley, 270 F.2d 396, 397, 123 USPQ 215, 216 (CCPA 1959)), which does not require that "an inventor or his attorney drop all other work and concentrate on the particular invention involved." Emery v. Ronden, 188 USPQ 264, 268 (Bd. Pat. Inter. 1974). See MPEP 2138.06.

As stated in the enclosed Rule 131 Declaration of Richard C. Auchterlonie, the applicants' patent attorney was concurrently working on a backlog of the following other related and unrelated cases for EMC Corporation:

Ser. 10/645,976 Filed 08-22-2003 Published as US 2005-0044080 A1

Ser. 10/646,851 Filed 08-22-2003 Published as US 2005-0044162 A1

Ser. 10/654,137 Filed 09-03-2003 Published as US 2005-0050107 A1

Ser. 10/655,693 Filed 09-05-2003 Published as US 2004-0054866 A1

Ser. 10/668,546 Filed 09-23-2003 Published as US 2005-0065986 A1

Ser. 10/668,783 Filed 09-23-2003 Published as US 2005-0065985 A1

These other published patent applications are cited on the enclosed IDS form.

The red-lined first draft of the patent application included many of the same or similar figures as the patent application as filed, and many of the same or similar claims as the patent application as filed, as summarized in the following table.

Patent Application as Filed	Red-lined first draft
FIG. 1	FIG. 1
FIG. 2	FIG. 2
FIG. 3	FIG. 3
FIG. 4	FIG. 4
FIG. 5	FIG. 5
FIG. 6	FIG. 6
FIG. 7	FIG. 50
FIG. 8	
FIG. 9	FIG. 5a
FIG. 10	FIG. 5b
FIG. 11	FIG. 7
FIG. 12	FIG. 8
FIG. 13	FIG. 9
FIG. 14	FIG. 10
FIG. 15	FIG. 11
FIG. 16	FIG. 12
FIG. 17	FIG. 13
FIG. 18	FIG. 14
FIG. 19	
...	
FIG. 36	

Claim 1	Claim 1
Claim 2	Claim 2
Claim 3	Claim 3
Claim 4	Claim 4
Claim 5	Claim 5
Claim 6	Claim 6
Claim 7	Claim 7
Claim 8	Claim 8
Claim 9	Claim 9
Claim 10	Claim 10
Claim 11	Claim 11
Claim 12	Claim 12
Claim 13	Claim 13
Claim 14	Claim 14
Claim 15	Claim 15
Claim 16	Claim 16
Claim 17	
...	
Claim 31	
Claim 32	Claim 17
Claim 33	Claim 18
Claim 34	Claim 19
Claim 35	Claim 20
Claim 36	Claim 21
Claim 37	Claim 22
Claim 38	Claim 23
Claim 39	Claim 24
Claim 40	Claim 25
Claim 41	Claim 26

Claim 42	
Claim 43	
Claim 44	Claim 27
Claim 45	Claim 28
Claim 46	Claim 29
Claim 47	Claim 30
Claim 48	Claim 31
Claim 49	Claim 32
Claim 50	Claim 33
Claim 51	
...	
Claim 57	
Claim 58	Claim 34
Claim 59	Claim 35
Claim 60	Claim 36
Claim 61	Claim 37

Support for the applicants' "network file server" being a computer is found, for example, in the red-lined first draft in FIG. 1 and on page 9 lines 19-21 and page 10 lines 7-9. Support for the file being in data storage is found, for example, in the red-lined first draft in FIG. 2 (file system 54 in cached disk array 29) as described on page 11 lines 10-16, page 12 lines 1-21, page 40 line 11, page 44 line 15, etc., and page 48 line 10-12.

Following is a mapping of the elements of applicants' claims to the page and line numbers in the red-lined first draft and the drawing reference numbers and figure numbers in the red-lined first draft:

Independent claim 1 defines a method of operating a network file server (21 in FIG. 1; page 9 lines 19-23) for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to a file (FIG. 10; page 24 lines 6-10). (Page 3, lines 19-21.) The method includes the network file server responding to a concurrent write request from a client by obtaining a lock for the file (step 101 in FIG. 7), and then preallocating a metadata block for the file (step 102 in FIG. 7), and then releasing the lock for the file (step 103 in FIG. 7); and then asynchronously writing to the file (step 104 in FIG. 7); and then obtaining the lock for the file (step 106 in FIG. 7); and then committing the metadata block to the file (step 107 in FIG. 7); and then releasing the lock for the file (step 108 in FIG. 7). (Page 3 line 21 to page 4 line 2; page 21 line 10 to page 22 line 7.)

Independent claim 13 defines a method of operating a network file server (21 in FIG. 1; page 9 lines 19-23) for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to a file (FIG. 10; page 24 lines 6-10). (Page 4, lines 3-4.) The method includes the network file server responding to a concurrent write request from a client by preallocating a block for the file (step 102 in FIG. 7); and then asynchronously writing to the file (step 104 in FIG. 7); and then committing the block to the file (step 107 in FIG. 7). (Page 4, lines 4-7; page 21 lines 11-19 and page 21 line 21 to page 22 line 4.) The asynchronous writing to the file includes a partial write to a new block (123 in FIG. 9) that has been copied at least in part from an original block (121 in FIG. 9) of the file. (Page 4, lines 7-9; page 23 lines 6-14.) The method includes checking a partial block conflict queue (73 in FIG. 4;

page 13 lines 11-15; page 14 lines 13-15) for a conflict with a concurrent write to the new block (step 151 in FIG. 13; page 28 lines 13-16), and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block (step 156 in FIG. 13; page 29 lines 4-8), and then performing the partial write to the new block (step 157 in FIG. 13; page 29 lines 8-11). (Page 4, lines 7-13.)

Applicants' independent 15 defines a method of operating a network file server (21 in FIG. 1; page 9 lines 19-23) for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to a file (FIG. 10; page 24 lines 6-10). (Page 4, lines 14-15.) The method includes the network file server responding to a concurrent write request from a client by preallocating a metadata block for the file (step 102 in FIG. 7), and then asynchronously writing to the file (step 104 in FIG. 7), and then committing the metadata block to the file (step 107 in FIG. 7). (Page 4, lines 15-18; page 21 lines 11-19 and page 21 line 21 to page 22 line 4.) The method further includes gathering together preallocated metadata blocks for a plurality of client write requests to the file (step 117 in FIG. 8), and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining a lock for the file (step 106 in FIG. 7), committing the gathered preallocated metadata blocks for the plurality of client write requests to the file (step 107 in FIG. 7; step 118 in FIG. 8), and then releasing the lock for the file (step 108 in FIG. 7). (Page 4, lines 18-23; page 21 line 21 to page 22 line 7; page 22 line 16 to page 23 line 5.)

Independent 25 defines a method of operating a network file server (21 in FIG. 1; page 9 lines 19-23) for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent read and write access (page 14 lines 10-12) to a file (FIG. 10; page 24 lines 6-10). The method includes the network file server responding to a concurrent write request from a client by preallocating a metadata block for the file (step 102 in FIG. 7), and then asynchronously writing to the file (step 104 in FIG. 7), and then committing the metadata block to the file (step 107 in FIG. 7). (Page 21 lines 11-19 and page 21 line 21 to page 22 line 4.) The network file server includes disk storage (29 in FIGS. 1 and 2; page 9 lines 18-21; page 11 lines 12-14) containing a file system (54 in FIG. 2 and FIG. 3; page 10 line 21 to page 11 line 2 and page 11 lines 12-14; page 12 lines 4-9), and a file system cache (51 in FIG. 2 and FIG. 3; page 11 lines 14-16; page 12 lines 1-3) storing data of blocks (134, 135, 138, 139 in FIG. 10; page 24 lines 6-10) of the file. The method includes the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (FIG. 5b; page 18 line 21 to page 19 line 14.) The method further includes the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale. (FIG. 5b; page 18 line 22 to page 19 line 5; page 19 line 15 to page 20 line 17 .)

Independent claim 32 defines a method of operating a network file server (21 in FIG. 1; page 9 lines 19-23) for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to a file (FIG. 10; page 24 lines 6-10). (Page 5, lines 1-2.) The method includes the network file server responding to a concurrent write request from a client by executing a write thread (FIG. 7). (Page 4, lines 3-4.). Execution of the write thread includes obtaining an allocation mutex for the file (step 101 in FIG. 7), and then preallocating new metadata blocks that need to be allocated for writing to the file (step 102 in FIG. 7), and then releasing the allocation mutex for the file (step 103 in FIG. 7); and then issuing asynchronous write requests for writing to the file (step 104 in FIG. 7), waiting for callbacks indicating completion of the asynchronous write requests (step 105 in FIG. 7), and then obtaining the allocation mutex for the file (step 106 in FIG. 7), and then committing the preallocated metadata blocks (step 107 in FIG. 7), and then releasing the allocation mutex for the file (step 108 in FIG. 7). (Page 4, lines 4-10; page 21 line 10 to page 22 line 7.)

Independent claim 33 defines a network file server (21 in FIG. 1; page 9 lines 19-23). (Page 5, line 11.) The network file server includes storage (29 in FIGS. 1 and 2; page 9 lines 18-21; page 11 lines 12-14) for storing a file (FIG. 10; page 24 lines 6-10), and at least one processor (26, 27, 28 in FIG. 1; page 9 lines 19-21) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to the file. (Page 5, lines 12-13.) The network file server is programmed for responding to a concurrent write request from a client by obtaining a lock for the file (step 101 in FIG. 7), and then preallocating a metadata block for the file (step 102 in FIG. 7), and then releasing the lock

for the file (step 103 in FIG. 7), and then asynchronously writing to the file (step 104 in FIG. 7), and then obtaining the lock for the file (step 106 in FIG. 7), and then committing the metadata block to the file (step 107 in FIG. 7), and then releasing the lock for the file (step 108 in FIG. 7). (Page 5, lines 13-18; page 21 line 10 to page 22 line 7.)

Independent claim 47 defines a network file server (21 in FIG. 1; page 9 lines 19-23). (Page 5, line 19.) The network file server includes storage (29 in FIGS. 1 and 2; page 9 lines 18-21; page 11 lines 12-14) for storing a file (FIG. 10; page 24 lines 6-10), and at least one processor (26, 27, 28 in FIG. 1; page 9 lines 19-21) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to the file. (Page 5, lines 20-21.) The network file server is programmed for responding to a concurrent write request from a client by preallocating a block for the file (step 102 in FIG. 7); and then asynchronously writing to the file (step 104 in FIG. 7); and then committing the block to the file (step 107 in FIG. 7). (Page 5, line 21 to page 6, line 1; page 21 lines 11-19 and page 21 line 21 to page 22 line 4.) The network file server includes a partial block conflict queue (73 in FIG. 4; page 13 lines 11-15; page 14 lines 13-15) for indicating a concurrent write to a new block that is being copied at least in part from an original block of the file. (Page 6, lines 1-3.) The network file server is programmed for responding to a client request for a partial write to the new block by checking the partial block conflict queue for a conflict (step 151 in FIG. 13; page 28 lines 13-16), and upon finding an indication of a conflict, waiting until resolution of the conflict with the concurrent write to the new block of the file (step 156 in FIG. 13; page 29 lines

4-8), and then performing the partial write to the new block of the file (step 157 in FIG. 13; page 29 lines 8-11). (Page 6, lines 3-7.)

Independent claim 49 defines a network file server (21 in FIG. 1; page 9 lines 19-23). (Page 6, line 8.) The network file server includes storage (29 in FIGS. 1 and 2; page 9 lines 18-21; page 11 lines 12-14) for storing a file (FIG. 10; page 24 lines 6-10), and at least one processor (26, 27, 28 in FIG. 1; page 9 lines 19-21) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to the file. (Page 6, lines 9-10.) The network file server is programmed for responding to a concurrent write request from a client by preallocating a metadata block for the file (step 102 in FIG. 7), and then asynchronously writing to the file (step 104 in FIG. 7), and then committing the metadata block to the file (step 107 in FIG. 7). (Page 6, lines 10-13; page 21 lines 11-19 and page 21 line 21 to page 22 line 4.) The network file server is programmed for gathering together preallocated metadata blocks for a plurality of client write requests to the file (step 117 in FIG. 8), and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining a lock for the file (step 106 in FIG. 7), committing the gathered preallocated metadata blocks for the plurality of client write requests to the file (step 107 in FIG. 7; step 118 in FIG. 8), and then releasing the lock for the file (step 108 in FIG. 7). (Page 6, lines 13-18; page 21 line 21 to page 22 line 7; page 22 line 16 to page 23 line 5.)

Independent 51 defines a network file server (21 in FIG. 1; page 9 lines 19-23). The network file server includes disk storage (29 in FIGS. 1 and 2; page 9 lines 18-21; page 11 lines

12-14) containing a file system (54 in FIG. 2 and FIG. 3), and a file system cache (51 in FIG. 2 and FIG. 3; page 11 lines 14-16; page 13 lines 8-16) storing data of blocks of a file (FIG. 10; page 24 lines 6-10) in the file system. (Page 6, lines 20-22.) The network file server is programmed for responding to a concurrent write request from a client (23, 24, 25 in FIG. 1; page 9 lines 18-19) by preallocating a metadata block for the file (step 102 in FIG. 7); and then asynchronously writing to the file (step 104 in FIG. 7), and then committing the metadata block to the file (step 107 in FIG. 7). (Page 21 lines 11-19 and page 21 line 21 to page 22 line 4.) The network file server is further programmed for responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (FIG. 5b; page 18 line 21 to page 19 line 14.) The network file server is programmed for responding to concurrent read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale. (FIG. 5b; page 18 line 22 to page 19 line 5; page 19 line 15 to page 20 line 17.)

Independent claim 58 defines a network file server (21 in FIG. 1; page 9 lines 19-23). (Page 6, lines 19-20.) The network file server includes storage (29 in FIGS. 1 and 2; page 9 lines 18-21; page 11 lines 12-14) for storing a file (FIG. 10; page 24 lines 6-10), and at least one processor (26, 27, 28 in FIG. 1; page 9 lines 19-21) coupled to the storage for providing clients

(23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to the file. (Page 6, lines 20-22.) The network file server is programmed with a write thread (FIG. 7) for responding to a concurrent write request from a client by obtaining an allocation mutex for the file (step 101 in FIG. 7), and then preallocating new metadata blocks that need to be allocated for writing to the file (step 102 in FIG. 7), and then releasing the allocation mutex for the file (step 103 in FIG. 7), and then issuing asynchronous write requests for writing to the file (step 101 in FIG. 7), waiting for callbacks indicating completion of the asynchronous write requests (step 101 in FIG. 7), and then obtaining the allocation mutex for the file (step 106 in FIG. 7); and then committing the preallocated metadata blocks (step 107 in FIG. 7), and then releasing the allocation mutex for the file (step 108 in FIG. 7). (Page 6, line 22 to page 7, line 6; page 21 line 10 to page 22 line 7.)

Independent claim 61 defines a network file server (21 in FIG. 1; page 9 lines 19-23). (Page 7, line 7.) The network file server includes storage (29 in FIGS. 1 and 2; page 9 lines 18-21; page 11 lines 12-14) for storing a file (FIG. 10; page 24 lines 6-10), and at least one processor (26, 27, 28 in FIG. 1; page 9 lines 19-21) coupled to the storage for providing clients (23, 24, 25 in FIG. 1; page 9 lines 18-19) with concurrent write access (page 14 lines 10-12) to the file. (Page 7, lines 8-9.) The network file server is programmed for responding to a concurrent write request from a client by preallocating a block for writing to the file (step 102 in FIG. 7), asynchronously writing to the file (step 104 in FIG. 7), and then committing the preallocated block (step 107 in FIG. 7). (Page 7, lines 9-12; page 21 lines 11-19 and page 21 line 21 to page 22 line 4.) The network file server also includes an uncached write interface (63 in

FIG. 3), a file system cache (51 in FIG. 2 and FIG. 3), and a cached read-write interface (61 in FIG. 3). (Page 7, lines 12-13; page 12 line 1 to page 13 line 10.) The uncached write interface bypasses the file system cache for sector-aligned write operations (FIG. 3; step 85 in FIG. 5), and the network file server is programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface (FIG. 6, steps 88 and 89). (Page 7, lines 13-16; page 12 lines 10-15; page 16 lines 3-6; page 16 line 19 to page 17 line 2.)

Dependent claims 2. and 34 further specify that the file (FIG. 10) further includes a hierarchy of blocks including an inode block (313 in FIG. 10) of metadata, data blocks (132, 134 in FIG. 10) of file data, and indirect blocks of metadata (133, 135 in FIG. 10), and wherein the metadata block for the file is an indirect block of metadata. (Page 27 lines 7-18; page 35 lines 14-16, page 41 lines 1-4.)

Dependent claims 3 and 35 further include copying data from an original indirect block of the file (121 in FIG. 9; 136 in FIG. 10 and FIG. 11) to the metadata block for the file (123 in FIG. 9; 141 in FIG. 12), the original indirect block of the file having been shared between the file and a read-only version of the file. (Page 14 lines 1-4, page 23 line 6 to page 25 line 14.)

Dependent claims 4 and 36 further include concurrent writing for more than one client to the metadata block for the file. (Page 14, lines 10-15.)

Dependent claims 5 and 37 further include asynchronous writing to the file including a partial write to a new block (123 in FIG. 9) that has been copied at least in part from an original block (121 in FIG. 9) of the file, and wherein the method further includes checking a partial block conflict queue (73 in FIG. 4; page 13 lines 11-15; page 14 lines 10-15) for a conflict with a concurrent write to the new block (step 151 in FIG. 13; page 28 lines 13-16), and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block (step 102 in FIG. 7), copying at least a portion of the original block of the file to the new block (step 153 in FIG. 13), and performing the partial write to the new block (step 154 in FIG. 13). (Page 27 lines 4-6, page 23 lines 6-14, page 28 line 13 to page 29 line 11.)

Dependent claims 6 and 38 are similar to claim 13 to the extent that claims 6 and 38 further define that the asynchronous writing to the file includes a partial write to a new block (123 in FIG. 9) that has been copied at least in part from an original block (121 in FIG. 9) of the file, and wherein the method further includes checking a partial block conflict queue (73 in FIG. 4; page 13 lines 11-15; Page 14 lines 10-15) for a conflict with a concurrent write to the new block (step 151 in FIG. 13; page 28 lines 13-16), and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block (step 156 in FIG. 13; page 29 lines 4-8), and then performing the partial write to the new block (step 157 in FIG. 13; page 29 lines 8-11).

Dependent claims 7, 14 and 39 further recite placing a request for the partial write in a partial write wait queue (74 in FIG. 4, step 156 in FIG. 13; page 28 lines 3-12) upon finding an

indication of a conflict with a concurrent write to the new block (step 151 in FIG. 13), and performing the partial write upon servicing the partial write wait queue (step 157 in FIG. 13). (Page 29 lines 4-11, page 36 lines 18-21, page 38 lines 18-21, page 42 line 22 to page 42 line 2.)

Dependent claims 8 and 40 further recite checking an input-output list (75 in FIG. 4; page 13 lines 11-15, page 14 lines 16-19) for a conflicting prior concurrent access to the file, and upon finding a conflicting prior concurrent access to the file, suspending the asynchronous writing to the file until the conflicting prior concurrent access to the file is no longer conflicting (step 87 in FIG. 5; page 16 lines 13-18). (Page 37 lines 1-5, page 43 lines 13-17.)

Dependent claims 9 and 41 further recite providing a sector-level granularity of byte range locking (step 85 in FIG. 5, step 87 in FIG. 5, step 88 in FIG. 9, 122 in FIG. 9, page 12 lines 10-21, page 13 lines 4-10, page 14 lines 10-15, page 16 lines 3-7, page and 19-21, page 23 lines 10-14, page 25 line 21 to page 26 line 2,) for concurrent write access to the file by the suspending of the asynchronous writing to the file until the conflicting prior concurrent access is no longer conflicting (step 156 in FIG. 13, step 157 in FIG. 13, page 29 lines 4-11) . (Page 33 lines 11-13, page 37 lines 7-10, page 43 lines 13-16.)

Dependent claims 10 and 44 further recite writing the metadata block to a log (55 in FIG 1 and FIG. 2) in storage (29 in FIG. 2) of the network file server for committing the metadata block for the file (step 107 in FIG. 7; page 11 lines 19-21, page 12 lines 6-9 and 17-21, page 14 lines 20-22, page 21 line 21 to page 22 line 1, page 37 lines 12-13, page 43 liens 19-21.)

Dependent claims 11 and 45 are similar to claim 15 to the extent that claims 11 and 45 further define gathering together preallocated metadata blocks for a plurality of client write requests to the file (step 117 in FIG. 8), and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file (step 106 in FIG. 7), committing the gathered preallocated metadata blocks for the plurality of client write requests to the file (step 107 in FIG. 7; step 118 in FIG. 8), and then releasing the lock for the file (step 108 in FIG. 7). (Page 4, lines 18-23; page 21 line 21 to page 22 line 7; page 22 line 16 to page 23 line 5.)

Dependent claims 12, 16 and 46 further include checking whether a previous commit is in progress (step 111 in FIG. 8) after asynchronously writing to the file (steps 103 and 104 in FIG. 7) and before obtaining the lock for the file for committing the metadata block to the file (step 112 or step 118 in FIG. 8), and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on a staging queue (76 in FIG. 4) for the file (step 117 in FIG. 8). (Page 22 lines 8-11 and page 22 line 16 to page 23 line 5.)

Dependent claim 17 is similar to claim 25 to the extent that claim 17 further defines the network file server responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (FIG. 5b; page 18 line 21 to page 19 line 14.)

Dependent claim 18 is similar to claim 25 to the extent that claim 18 further defines the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale. (FIG. 5b; page 18 line 22 to page 19 line 5; page 19 line 15 to page 20 line 17.)

Dependent claims 19, 26 and 52 further include the network file server checking a read-in-progress flag for a file block (FIG. 5b; cf. step 510 in FIG. 10 of the application as filed) upon finding that the file block is not in the file system cache (FIG. 5b; cf. step 92 in FIG. 10 of the application as filed), and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage (FIG. 5b; cf. step 511 in FIG. 10 of the application as filed), and then again checking whether the file block is in the file system cache (FIG. 5b, cf. step 92 in FIG. 10 of the application as filed). (Page 18 line 22 to page 19 line 5; page 19 line 15 to page 20 line 17.)

Dependent claims 20, 27 and 53 further includes the network file server setting a read-in-progress flag for a file block (FIG. 5b, cf. step 512 in FIG. 10 of the application as filed) upon finding that the file block is not in the file system cache (FIG. 5b, cf. step 92 in FIG. 10 of the

application as filed) and then beginning to read the file block from the file system in disk storage FIG. 5b, cf. step 512 in FIG. 10 of the application as filed), clearing the read-in-progress flag upon writing to the file block on disk (FIG. 5b, cf. steps 515 and 516 in FIG. 10 of the application as filed), and inspecting the read-in-progress flag (FIG. 5b, cf. step 513 in FIG. 10 of the application as filed) to determine whether the file block has become stale due a concurrent write to the file block. (Page 18 line 22 to page 19 line 5; page 19 line 15 to page 20 line 17.)

Dependent claim 21, 28 and 54 further include the network file server maintaining a generation count (FIG. 5b, cf. step 512 in FIG. 10 of the application as filed) for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache (FIG. 5b, cf. step 92 in FIG. 10 of the application as filed), and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block (FIG. 5b, cf. step 513 in FIG. 10 of the application as filed). (Page 18 line 22 to page 19 line 5; page 19 line 15 to page 20 line 17.)

Dependent claim 22 further recites processing multiple concurrent read and write requests by pipelining the requests through a first processor and a second processor, the first processor performing metadata management for the multiple concurrent read and write requests, and the second processor performing asynchronous reads and writes for the multiple concurrent read and write requests. (FIG. 50, page 17 line 3 to page 18 line 18.)

Dependent claim 23 further recites serializing the reads by delaying access for each read to a block that is being written to by a prior, in-progress write until completion of the write to the block that is being written to by the prior, in-progress write. (Step 91 in Fig. 5a, Fig. 5b, page 18 line 19 to page 19 line 5, page 19 line 15 to page 20 line 17.)

Dependent claim 24 further recites serializing the writes by delaying access for each write to a block that is being accessed by a prior, in-progress read or write until completion of the read or write to the block that is being accessed by the prior, in-progress read or write. (Steps 82, 86, 87 in FIG. 5; Fig. 5b,; page 15 line 15 to page 16 line 18, page 18 line 22 to page 20 line 17.)

Dependent claim 42 is similar to claim 8 in that is further recites maintaining an input-output list (75 in FIG. 4; page 13 lines 11-15, page 14 lines 16-19) of concurrent reads and writes to the file, and when reading from the file, for checking the input-output list for a conflicting prior concurrent write access to the file, and upon finding a conflicting prior concurrent write access to the file, suspending the reading to the file until the conflicting prior concurrent write access to the file is no longer conflicting. (Step 87 in FIG. 5; page 16 lines 13-18, page 37 lines 1-5, page 43 lines 13-17.)

Dependent claim 43 is similar to claim 9 and further recites that the network file server as is further programmed so that the suspending of the reading to the file until the conflicting prior

concurrent write access to the file is no longer conflicting provides a sector-level granularity of byte range locking for concurrent read access to the file. (Step 81 in FIG 5a, FIG. 5b, page 14 lines 15-19, page 18 line 19 to page 19 line 5, page 19 line 13 to page 20 line 17, page 43 lines 13-17,)

Dependent claim 50 further recites checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file, and upon finding that a previous commit is in progress, placing a request for committing the metadata block to the file on a staging queue (76 in FIG. 4; step 117 in FIG. 8) for the file. (Page 13 lines 11-15, page 14 line 20 to page 15 line 10, page 22 lines 16-21, page 46 lines 2-6.)

Dependent claim 59 is similar to claim 61 to the extent that claim 59 further defines an uncached write interface (63 in FIG. 3), a file system cache (51 in FIG. 2 and FIG. 3) and a cached read-write interface (61 in FIG. 3; page 7, lines 12-13; page 12 line 1 to page 13 line 10) wherein the uncached write interface bypasses the file system cache for sector-aligned write operations. (FIG. 3; step 85 in FIG. 5; page 12 lines 10-16; page 16 lines 3-6.)

Dependent claim 60 is similar to claim 61 to the extent that claim 60 defines that the network file server is further programmed to invalidate cache blocks in the file system cache including sectors being written to by the uncached write interface. (FIG. 6, steps 88 and 89; page 16 line 19 to page 17 line 2.)

Dependent 62, 63, 64, 65, 66, and 67 further recite a final step of returning to said client an acknowledgement of the writing to the file (step 116 in FIG. 8; page 15 lines 11-14, page 22 lines 14-15).

Dependent claims 68, 69, 70, 71, 72, and 73 further recite a final step of saving the file in disk storage of the network file server. (Page 11 lines 19-21, page 23 line 21 to page 24 line 5, page 33 line 22 to page 34 line 2).

2b. Chang et al. does not show every element arranged as in applicants' claims.

"For a prior art reference to anticipate in terms of 35 U.S.C. § 102, every element of the claimed invention must be identically shown in a single reference." Diversitech Corp. v. Century Steps, Inc., 7 U.S.P.Q.2d 1315, 1317 (Fed. Cir. 1988), quoted in In re Bond, 910 F.2d 831,15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990) (vacating and remanding Board holding of anticipation; the elements must be arranged in the reference as in the claim under review, although this is not an *ipsis verbis* test).

As amended, applicants' claim 1 calls for a network file server computer responding to a concurrent write request from a client by performing a sequence of seven steps. The network file server computer responds by:

(a) obtaining a lock for the file and then

(b) preallocating a metadata block for the file; and then

(c) releasing the lock for the file; and then

(d) asynchronously writing to the file in the data storage; and then

(e) obtaining the lock for the file; and then

(f) committing the metadata block to the file in the data storage; and then

(g) releasing the lock for the file.

The applicants' drawings show these steps (a) to (g) in FIG. 11 in boxes 101, 102, 103, 104-105, 1-6, 107, and 108, respectively, as described in applicants' specification on page 27 lines 3-23.

In applicants' view, Chang does not disclose a network file server computer responding to a concurrent write request from a client by performing the applicants' claimed sequence of seven steps. Instead, Chang discloses a network file server computer that responds to concurrent write requests by performing different sequences of steps, depending on whether or not the requested access is a read (step 520 in Chang's FIG. 5), the requested access is a write to a file that allows concurrent readers and writers (step 530 in FIG. 5), or the request is a write requiring change in allocation (sep 550 in FIG. 5). (Chang, paragraphs [0060], [0061], and [0062].)

In the applicants' claim 1, the network file server computer is responding to a concurrent write request by, among other things, preallocating a metadata block for the file, and committing the metadata block to the file in the data storage. Therefore, in the context of Chang, this applicants' concurrent write request is "a write requiring a change in allocation" because in response to the request a process is allocating a block for the file. (See Chang step 550 in FIG. 5; Chang paragraphs [0015], [0047].) In this case, in Chang FIG. 5, execution branches from step 550 to step 540 to "take writer lock" as described in Chang paragraphs [0062] and [0063]. When a writer lock is taken, other processes (e.g. 440, 450, 460, 470 in Chang FIG. 4B), writers and

readers, will not be allowed to proceed and instead other processes attempting to access the same file will "spin on the lock" prior to obtaining a writer lock (step 540 in Chang FIG. 5) or a reader lock (step 560 in Fig. 5) on the same file. See Chang paragraph [0049]. A read or write lock must be obtained to gain access to the file. See Chang paragraphs [0017], [0021], [0046] ("The present invention does not avoid or bypass the file locking, but makes use of the file locks to permit multiple concurrent readers and writers."), and [0049]. Therefore, a person of ordinary skill in the art would understand that when responding to a concurrent write request that would add a block to a file, the network file server computer of Chang would obtaining a lock for the file, add to the file a block to which data is written, and then release the lock for the file. Chang, however, does not disclose how a block to which data is written is added to the file. Therefore, applicants respectfully submit that Chang does not disclose the applicants' recited sequence of the six steps (b), (c), (d), (e), (f), and (g).

Page 4 of the Official Action suggests that Chang paragraph [0048] discloses applicants' step b) of preallocating a metadata block for the file. However, paragraph [0049] mentions that a case of a write operation that is being performed by a process is one that requires or results in a change in allocation of data blocks of the file 400. Although Chang paragraph [0015] further discloses that a change in allocation of data blocks of a file may include allocating a block of data, Chang does not disclose how a block of data is allocated to the file. For example, there is nothing in Chang disclosing that a metadata block is preallocated for the file, and then the file is asynchronously written to, and then the metadata block is committed to the file in data storage, as recited in applicants' claim 1.

Page 4 of the Official Action suggests that Chang paragraphs [0049] and [0054] disclose applicants' step c) of releasing the flock for the file, and Chang paragraph [0056] discloses applicants' step g) of releasing the lock for the file. However, paragraphs [0049], [0054], and [0059] do not give details as to how or when a read or write lock on the file is released. In particular, there is nothing in Chang to suggest that the network file server computer should release the lock on the file more than once when responding to a concurrent write request from a client. Instead, Change teaches away from releasing the lock for the file in step (c) and then asynchronously writing to the file in step (d) and then again obtaining the lock for the file in step (e), because Chang, as set out above, teaches that the file would be write-locked when a write operation adding a block to the file is performed on the file.

Page 4 of the Official Action suggests that Chang paragraph [0048] discloses applicants' step a) of obtaining a lock for the file, and paragraph [0054] discloses applicants' step e) of obtaining the lock for the file. However, there is nothing in Chang to suggest that the network file server computer should obtain a lock on the file more than once when responding to a concurrent write request from a client. Instead, as discussed above, Chang teaches that the file would be write-locked when a write operation adding a block to the file is performed on the file, and in this case the file would not be accessed until the write lock is obtained. In other words, for responding to a concurrent write request, Chang teaches a first step of obtaining a write-lock for the file and then performing the requested write access to the file and then releasing the write-lock.

With respect to applicants' dependent claim 2, page 4 of the Official Action cites Chang paragraphs [0047]-[0048]. Although paragraph [0051] refers to an update to the metadata

structure associated with the file, Chang does not disclose details of this metadata structure such as an indirect block of metadata.

With respect to applicants' dependent claim 4, page 5 of the Official Action cites Chang paragraphs [0047]-[0048]. Although paragraph [0051] refers to an update to the metadata structure associated with the file, Chang does not disclose details of this metadata structure such as copying data from an original indirect block of the file to the metadata block for the file, the original indirect block of the file having been shared between the file and a read-only version of the file.

With respect to applicants' dependent claim 10, page 5 of the Official Action cites Chang paragraph [0056]. Although paragraph [0056] refers to software based mechanisms, such as database application serialization mechanisms, for serializing concurrent write operations for maintaining file structure integrity, Chang does not teach writing a metadata block to a log in storage of the network file server for committing the metadata block.

With respect to applicants' dependent claim 11, page 5 of the Official Action cites Chang paragraphs [0047]-[0056]. Chang, however, does not disclose gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file. Instead, as discussed above, if a client write request requires a change in allocation for the file (step 550 in Chang FIG. 5), then the file server takes a writer lock (step 550 in Chang) and this writer lock prevents other processes from taking a writer lock or reader lock. Therefore Chang teaches away from

gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

With respect to applicants' independent claim 15, as discussed above with reference to applicants' claim 11, it is respectfully submitted that Chang fails to disclose gathering together preallocated metadata blocks for a plurality of client write requests to the file, and committing together the preallocated metadata blocks for the plurality of client write requests to the file by obtaining the lock for the file, committing the gathered preallocated metadata blocks for the plurality of client write requests to the file, and then releasing the lock for the file.

With respect to applicants' independent claim 32, see applicants' remarks above with reference to applicants' claim 1. In short, Chang does not disclose how a block is added to the file. Therefore, applicants respectfully submit that Chang does not disclose that new metadata bocks are added to the file by the applicants' recited sequence of the six steps (b), (c), (d), (e), (f), and (g).

With respect to claims 33-36, applicants respectfully traverse for the reasons given above with reference to claims 1-4.

With respect to claims 44-45, applicants respectfully traverse for the reasons given above with reference to claim 10-11.

With respect to independent claim 49, applicants respectfully traverse for the reasons given above with reference to claim 15.

With respect to independent claim 58, applicants respectfully traverse for the reasons given above with reference to claim 32.

With respect to independent claim 61, Chang mentions a file buffer cache in paragraph [0044] and Chang mentions certain alignment and length restrictions in paragraph [0052]. Chang, however, does not disclose that the network file server is further programmed to invalidate cache blocks in a file system cache including sectors being written to by an uncached write interface.

3. Reply to the Rejections under 35 U.S.C. 103(a)

In paragraph 8 on page 8 of the Official Action, claims 5-9, 12-14, 16-21, 23-28, 37-43, 46-48, and 50-54 were rejected under 35 U.S.C. 103(a) as being unpatentable over Chang in view of Marcotte U.S. 6,449,614. In response, applicants respectfully traverse. In short, as discussed above, Chang should be withdrawn as a reference in view of applicants' showing of prior invention. In addition, as discussed above, various elements of the respective base claims are missing from Chang, and Marcotte does not disclose these elements missing from Chang, so that the applicants' claimed invention does not result from the proposed combination of Chang and Marcotte. Moreover, the applicants' claims define a substantial improvement over Chang and Marcotte by providing a new way of handling a write request that changes the allocation of the data blocks to a file, and this new way of handling such a write request solves a long-felt

need for reducing contention during multi-threaded or multi-processor access to a shared file system.

Chang itself is evidence of a long-felt but unsolved problem that in a shared or parallel file system, the potential speed at which an application may execute is impaired by the need for file locking. Chang teaches that for "an application having its own serialization or locking mechanisms" (Chang, paragraph [0014]), "multiple processes may write to the same block of data within the file at approximately the same time as long as they are not changing the allocation of the block of data, i.e. either allocating the block, deallocating the block of data, or changing the block of data." (Chang, paragraph [0015].) The applicants' invention further solves this problem by enabling multiple processes to write to the file at approximately the same time when updating the metadata structure associated with the file. The applicants are citing additional evidence on the enclosed IDS form showing that shared or parallel file systems and their associated problems have been known for at least a decade prior to the filing of the applicants' patent application.

Marcotte discloses an interface system and methods for asynchronously updating a share resource with locking facility. Tasks make updates requested by calling tasks to a shared resource serially in a first come first served manner, atomically, but not necessarily synchronously, such that a current task holding an exclusive lock on the shared resource makes the updates on behalf of one or more calling tasks queued on the lock. Updates waiting in a queue on the lock to the shared resource may be made while the lock is held, and others deferred

for post processing after the lock is released. Some update requests may also, at the calling application's option, be executed synchronously. Provision is made for nested asynchronous locking. Data structures (wait_elements) describing update requests may queued in a wait queue for update requests awaiting execution by a current task, other than the calling task, currently holding an exclusive lock on the shared resource. Other queues are provided for queuing data structures removed from the wait queue but not yet processed; data structures for requests to unlock or downgrade a lock; data structures for requests which have been processed and need to be returned to free storage; and data structures for requests that need to be awakened or that describe post processing routines that are to be run while the lock is not held. (Abstract.)

With respect to applicants' dependent claims 5-9, 12, 16-21, 23-24, 37-43, and 50-54, these claims depend from the independent claims 1, 15, 33, and 49. Chang has been distinguished above with respect to the independent claims 1, 15, 33, and 49, and Marcotte does not provide the limitations of these independent claims that are missing from Chang. Therefore the dependent claims 5-9, 12, 16-21, 23-24, 37-43, and 50-54 are patentable over the proposed combination of Chang and Marcotte. It is respectfully submitted that it would not have been obvious for one of ordinary skill to combine Chang and Marcotte in the fashion as proposed in the final Official Action and then modify that combination by adding the missing limitations.

Applicants' dependent claim 5 adds to claim 1 the limitations of "wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking

a partial block conflict queue for a conflict with a concurrent write to the new block, and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing the partial write to the new block." Applicants' partial block conflict queue 73 is shown in applicants' FIG. 4 and described in applicant's specification on page 15 lines 17-22 as follows:

> The preallocation method allows concurrent writes to indirect blocks within the same file. Multiple writers can write to the same indirect block tree concurrently without improper replication of the indirect blocks. Two different indirect blocks will not be allocated for replicating the same indirect block. The write threads use the partial block conflict queue 73 and the partial write wait queue 74 to avoid conflict during partial block write operations, as further described below with reference to FIG. 13.

See also applicants' FIG. 13 and specification page 28 line 22 to page 29 line 7; and FIG. 17 and page 34 line 7 to page 35 line 5.

With respect to applicants' dependent claim 5, pages 8-9 of the Official Action recognizes that Chang fails to explicitly recite the limitations added by dependent claim 5, and says that Marcotte teaches these limitations, citing column 13, lines 35 through col. 14, line 43. However, Marcotte column 13, lines 35 through col. 14, line 43 deals with managing an I/O device holding queue above a device for queuing pending I/O's if the number of I/O's issued to the device exceeds a threshold that is adjustable by an application. It is not seen where Marcotte discloses a partial write to a new block that has been copied at least in part from an original

block of the file. Nor is it seen where Marcotte discloses checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon failing to find an indication of a conflict with a concurrent write to the new block, preallocating the new block, copying at least a portion of the original block of the file to the new block, and performing the partial write to the new block.

"[R]ejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." In re Kahn, 441 F. 3d 977, 988 (Fed. Cir. 2006). A fact finder should be aware of the distortion caused by hindsight bias and must be cautious of arguments reliant upon ex post reasoning. See KSR International Co. v. Teleflex Inc., 550 U.S. __, 82 USPQ2d 1385 (2007)), citing Graham, 383 U. S. at 36 (warning against a "temptation to read into the prior art the teachings of the invention in issue" and instructing courts to "guard against slipping into the use of hindsight.").

Applicants' claim 6 is similar to claim 5 in that it also adds to claim 1 the limitations of wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method further includes checking a partial block conflict queue for a conflict with a concurrent write to the new block. With respect to dependent claim 6, pages 9-10 of the Official Action also recognizes that Chang fails to explicitly recite the limitations added by dependent claim 6, and says that Marcotte teaches these limitations, again citing column 13, lines 35 through col. 14, line 43. However, Marcotte column 13, lines 35 through col. 14, line 43 deals with managing an I/O device holding queue above a device for queuing pending I/O's if the number of I/O's issued to the device

exceeds a threshold that is adjustable by an application. It is not seen where Marcotte discloses a partial write to a new block that has been copied at least in part from an original block of the file. Nor is it seen where Marcotte discloses checking a partial block conflict queue for a conflict with a concurrent write to the new block.

With respect to applicants' dependent claim 12, applicants respectfully submit that Chang does not further teach checking whether a previous commit is in progress after asynchronously writing to the file and before obtaining the lock for the file for committing the metadata block to the file. As discussed above with reference to applicants' claim 1, in response to a concurrent write request from a client, Chang does not obtain a lock for the file after the asynchronously writing to the file. Nor dos Marcotte column 12, line 7 through column 13, line 32 disclose these limitations missing from Chang or specifically deal with committing "metadata blocks" to a file.

Applicants' independent claim 13 recites "wherein the asynchronous writing to the file includes a partial write to a new block that has been copied at least in part from an original block of the file, and wherein the method includes checking a partial block conflict queue for a conflict with a concurrent write to the new block, and upon finding an indication of a conflict with a concurrent write to the new block, waiting until resolution of the conflict with the concurrent write to the new block, and then performing the partial write to the new block." Thus, Applicants' independent claim 13 is patentable over Chang in combination with Marcotte for the reasons given above with reference to applicants' claim 5.

Applicants' dependent claim 14 is dependent upon claim 13 and therefore is also patentable over Chang in combination with Marcotte for the reasons given above with reference to applicants' claim 13.

Applicants' dependent claim 16 adds to claim 15 the express limitations of claim 12 and therefore is patentable over Chang in combination with Marcotte for the reasons given above with reference to applicants' claims 12 and 15.

Applicants' dependent claim 17 adds to claim 15 the limitations of "wherein the network file server includes disk storage containing a file system, and a file system cache storing data of blocks of the file, and the method further includes the network file server computer responding to concurrent write requests by writing new data for specified blocks of the file to the disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache." Pages 14-15 of the Official Action recognizes that Chang fails to explicitly recite these limitations, and cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 18 adds to claim 17 the limitations of "the network file server computer responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become stale, and not writing to the file system cache a file block that has become stale." Page 15 of the Official Action again cites Marcotte column 12 line

7 through column 13, line 32. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 19 adds to claim 18 the limitations of "the network file server computer checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage, and then again checking whether the file block is in the file system cache." Page 16 of the Official Action again cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 20 adds to claim 18 the limitations of "the network file server computer setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block." Page 16 of the Official Action again cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 21 adds to claim 18 the limitations of "the network file server computer maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system

cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been read from the file system is the same as the generation count for the last read request for the same file block." Page 16 of the Official Action again cites Marcotte column 12 line 7 through column 13, line 32. However, it is not understood how the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

With reference to applicants' independent claim 25, page 18 of the Official Action recognizes that Chang fails to explicitly recite the network file server responding to concurrent write requests by writing new data for specified blocks of the file to disk storage without writing the new data for the specified blocks of the file to the file system cache, and invalidating the specified blocks of the file in the file system cache. (See, e.g., applicants' FIG. 10, steps 515 and 516; applicants' spec., page 23 lines 19-23 and page 24 lines 7-15.) Page 18 of the Official Action further recognizes that Chang fails to explicitly recite the network file server responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become state, and not writing to the file system cache a file block that has become stale. (See, e.g., applicants' FIG. 10, steps 92, 97, 513, 98; applicants' spec., page 23 lines 5-17; page 24 lines 16-22.) Page 19 of the Official Action cites Marcotte col. 12 line 7 through column 12, line 32 for all of these limitations not explicitly recited in Chang. However, it is not understood how all of the applicants' specific claim limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Marcotte column 12 line 7 through column 13, line 32, deals generally with maintaining a list of lock waiters. As shown in Marcotte FIG. 9, when a task waits for a lock, it queues its WAIT_ELEMENT to the lock using lock or wait routine 175 and also adds it WAIT_ELEMENT to a global list or queue 230 before it waits, and removes it from the global list 230 after it waits. As shown in Marcotte FIG. 11, do_wait 240 is executed each time a thread needs to go into a wait for a lock. Step 241 executes a lock_UpdateResource procedure. Step 242 waits on ecb; and upon receiving it, step 243 executes the lock_Update Resource procedure. Marcotte says that in this way, the task waiting on a lock 100 will only actually suspend itself on the WAIT call. Adding and removing tasks (WAIT ELEMENTS) from global list 230 is done in a manner guaranteed to make the calling task wait. FIG. 12 shows that an add_to_global routine 245 (with waitp-arg) includes step 246 which determines if prevent flag is null; if so, step 248 posts an error code in the ecb field 108 of the WAIT_ELEMENT being processed; and, if not, step 247 adds the WAIT_ELEMENT (in this case, task 232) to the head of global list 230. FIG. 13 shows a remove_from_global routine 250 (with waitp=arg) including step 251 which determines if prevent flag 124 is null. If so, return code (rc) is set to zero; and if not, this WAIT_ELEMENT (say, 233) is removed from global list 230. In step 254, the return code (rc) is returned to the caller. FIG. 14 shows a return_wait routine 260 (with waitp=prc) including step 261 which determines if waitp is null. If not, the WAIT_ELEMENT pointed to by waitp is returned to free storage. Return wait 260 is the post processing routine for remove_from_global 250, and remove_from_global communicates the address of the WAIT_ELEMENT via its return code, that is input to return_wait (automatically by the resource update facility) as prc. Return_wait 260 returns the WAIT_ELEMENT to free storage. Since routine 260 is a post processing routine,

the free storage return is NOT performed while holding the lock. This shows the benefits of a post processing routine, and passing the return value from a resource update routine to the post processing routine. FIG. 15 shows quiesce_global 265 wakes up all waiters and in step 267 tells them that the program is terminating due to error by way of prevent flag 124 being set to 1 in step 266. In step 268 pointer 231 and 234 are cleared so global list 230 is empty.

Thus, it is not understood how Marcotte's general teaching of a way of maintaining a list of lock waiters discloses the applicants' specific limitations of the network file server responding to concurrent write requests by writing new data for specified blocks of the file to disk storage without writing the new data for the specified blocks of the file to the file system cache, invalidating the specified blocks of the file in the file system cache, and responding to read requests for file blocks not found in the file system cache by reading the file blocks from the file system in disk storage and then checking whether the file blocks have become stale due to concurrent writes to the file blocks, and writing to the file system cache a file block that has not become state, and not writing to the file system cache a file block that has become stale.

"[R]ejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." In re Kahn, 441 F. 3d 977, 988 (Fed. Cir. 2006). A fact finder should be aware of the distortion caused by hindsight bias and must be cautious of arguments reliant upon ex post reasoning. See KSR International Co. v. Teleflex Inc., 550 U.S. __, 82 USPQ2d 1385 (2007)), citing Graham, 383 U. S. at 36 (warning against a "temptation to read into the prior art the teachings of the invention in issue" and instructing courts to "guard against slipping into the use of hindsight.").

Applicants' dependent claim 26 adds to claim 25 the limitations of "the network file server computer checking a read-in-progress flag for a file block upon finding that the file block is not in the file system cache, and upon finding that the read-in-progress flag indicates that a prior read of the file block is in progress from the file system in the disk storage, waiting for completion of the prior read of the file block from the file system in the disk storage, and then again checking whether the file block is in the file system cache." Page 19 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 27 adds to claim 25 the limitations of "the network file server computer setting a read-in-progress flag for a file block upon finding that the file block is not in the file system cache and then beginning to read the file block from the file system in disk storage, clearing the read-in-progress flag upon writing to the file block on disk, and inspecting the read-in-progress flag to determine whether the file block has become stale due a concurrent write to the file block." Page 20 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

Applicants' dependent claim 28 adds to claim 25 the limitations of "the network file server computer maintaining a generation count for each read of a file block from the file system in the disk storage in response to a read request for a file block that is not in the file system cache, and checking whether a file block having been read from the file system in the disk storage has become stale by checking whether the generation count for the file block having been

read from the file system is the same as the generation count for the last read request for the same file block." Page 20 of the Official Action again cites Marcotte column 12 line 7 through column 12, line 21. However, it is not understood how these specific limitations are disclosed in Marcotte column 12 line 7 through column 13, line 32.

With respect to dependent claim 37-38, applicants further traverse for the reasons given above with reference to claims 5-6.

With respect to dependent claims 46-48, applicants further traverse for the reasons given above with reference to claims 12-14.

With respect to dependent claim 50, applicants further traverse for the reasons given above with reference to claim 16.

With respect to claims 51-54, applicants respectfully traverse for the reasons given above with reference to claims 25-28.

On page 21 of the Official Action, applicants' dependent claims 59 and 60 were rejected under 35 U.S.C. 103(a) as being unpatentable over Chang in view of Xu et al. U.S. Pat. 6,324,581. In reply, applicants respectfully submit that applicants' dependent claims 59 and 60 are patentable over the proposed combination of Chang and Xu due to the limitations of their independent base claim 58. Chang is distinguished from the base claim 58 as discussed above with reference to claim 1. Xu is distinguished from the base claim 58 in a similar fashion Xu fails to disclose appellants' step c) of releasing the allocation mutex of the file [prior to the step d) of issuing asynchronous write requests for writing to the file], and appellants' step f) of

obtaining the allocation mutex for the file [after the step d) of issuing asynchronous write requests for writing to the file].

In view of the above, it is respectfully submitted that the application is in condition for allowance. Reconsideration and early allowance are earnestly solicited.

Respectfully submitted,

/ Richard C. Auchterlonie /

Richard C. Auchterlonie, Reg. No. 30,607
NOVAK DRUCE & QUIGG, LLP
1000 Louisiana, 53rd Floor
Houston, TX 77002
Telephone 713-571-3460
Telefax 713-456-2836
Richard.Auchterlonie@novakdruce.com